DAW-Integrated Beat Tracking for Music Production

Brett Dalton
University of Victoria
`brettdalton.ca@gmail.com`

David Johnson
University of Victoria
`davidjo@uvic.ca`

George Tzanetakis
University of Victoria
`gtzan@uvic.ca`

ABSTRACT

Rhythm analysis is a well researched area in music information retrieval that has many useful applications in music production. In particular, it can be used to synchronize the tempo of audio recordings with a digital audio workstation (DAW). Conventionally this is done by stretching recordings over time, however, this can introduce artifacts and alter the rhythmic characteristics of the audio. Instead, this research explores how rhythm analysis can be used to do the reverse by synchronizing a DAW's tempo to a source recording. Drawing on research by Percival and Tzanetakis, a simple beat extraction algorithm was developed and integrated with the Renoise DAW. The results of this experiment show that, using user input from a DAW, even a simple algorithm can perform on par with popular packages for rhythm analysis such as BeatRoot, IBT, and aubio.

1. INTRODUCTION

Tempo is a feature of audio which is commonly analyzed in Music Information Retrieval (MIR) due to its fundamental role in music. Tempo is described by a pulse, a set of steady intervals of time which govern the way that music is perceived and expressed. In a written piece of music, notes and rhythmic events are aligned in relation to this pulse, which is grouped and subdivided in various ways, ultimately forming the piece's structure. This underlying structure is necessary for musicians to synchronize with each other while performing, and it allows listeners to understand what they hear. The importance of tempo in music makes for a broad range of practical applications for its analysis, ranging from genre classification to DJ software, [1], [2], [3].

In digital music production, tempo is most often described by a piece's global beat rate, commonly measured in Beats Per Minute (BPM). Generally this is understood as a constant value which does not vary over the duration of the piece. This is different from how music is traditionally performed by live musicians, where the tempo will naturally drift without mechanically perfect synchronization to some clock.

This presents three possibilities for digital music producers trying to work with live source material: One, to synchronize the recording with a constant BPM by using time-stretching; two, to extract the structure from the original piece and synchronize their software to the recording; or, three, to work with the musical structure by ear, without digital assistance.

In order to explore the second option, this paper uses the work of Percival and Tzanetakis [4] with some modifications and additions, to demonstrate the effectiveness of beat tracking in a modern music production setting. This has been accomplished by developing a piece of software which integrates a simple beat extraction algorithm in a Digital Audio Workstation (DAW) to accomplish tasks which would otherwise be cumbersome when working with live recorded material.

The DAW that has been chosen for this research is Renoise, as it has a native scripting API that can be used to modulate tempo over time. The core algorithm was written and tested in Python, and Lua was used to integrate the code with Renoise's scripting interface. Later, the algorithm was ported to C for real-time use. The results of this work have been evaluated using mir-eval [1] [5], a python library designed for gathering statistics for common MIR tasks. The results are compared to the established beat tracking methods aubio [6], IBT [7], and BeatRoot, [8].

2. BACKGROUND

Beat tracking is a well researched area in MIR that has been analyzed in different contexts using a variety of methods. Early approaches to beat tracking involved processing the symbolic representations of music such as musical scores and MIDI data. However, as computing technology and theory developed, it became possible to analyze raw audio recordings for the purpose of beat extraction and the extraction of more complex metric information, [9], [10].

As noted by Gkiokas et al., most tempo and beat extraction algorithms share some common structural elements, [11]. The most common approach for processing an audio signal involves retrieving what is known as an onset strength signal (OSS), novelty curve, or salience function, [2], [12], [13]. This is a transformation of the original audio that attempts to capture a continuous function of rhythmic importance over time. A number of different metrics can be used to derive the OSS such as spectral flux, phase deviation, and complex domain methods, [14] as well as machine learning [15]. Peaks in this signal can be thought of as discrete rhythmic events that can be used in further analysis to extract information using a variety of techniques. These techniques may involve multi-agent

Copyright: © 2019 Brett Dalton et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 Unported License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

[1] `https://github.com/craffel/mir_eval`

systems [16], autocorrelation [4], and it is also common to use higher level musical features such as chord changes and drum patterns to obtain more accurate results, [10].

As of today, few DAWs incorporate true beat tracking for recorded material, but instead offer tools for manually solving tempo related problems such as audio to MIDI conversion, BPM detection, time stretching, and quantization. This is perhaps due to beat tracking being less reliable and less flexible than the alternatives.

There has been past work in creating an interactive system for beat tracking. In 2001, Dixon developed graphical software for beat extraction, data editing, sonification, and a variety of other tasks, [17]. At the time, he noted some deficiencies in similar tools; one being that they did not allow for the user to correct mistakes in the beat tracking process. The goal of creating this beat tracking software is to continue on Dixon's line of work and integrate a graphical user interface with a beat tracking system that is useful for real world applications in a music production environment.

3. ALGORITHM DESCRIPTION

Our proposed beat extraction algorithm contains the following stages similar to other methods: extract an onset strength signal from the desired audio file; extract individual onsets and their attributes such as loudness and timing with peak picking; and iterate over these onsets to find potential beats using a simple heuristic induction method.

The Streamlined tempo induction algorithm developed by Percival and Tzanetakis is designed with the intention of using the simplified forms of common techniques while still attaining reliable performance [4]. The task of tempo induction has different requirements from beat extraction, but they are naturally related. Our algorithm employs the same OSS and peak picking methods as the Streamlined algorithm, with some modifications and additions.

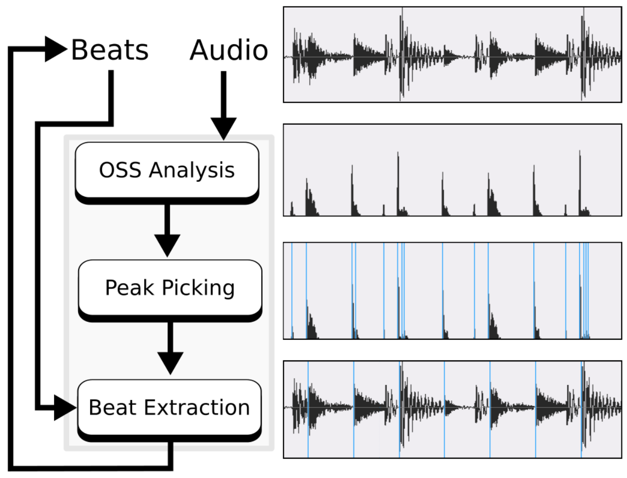


Figure 1. The proposed block diagram of the beat tracking system. Note the ability to generate, modify, and reprocess beats.

3.1 OSS Calculation

OSS is normally calculated using the spectral flux of the input signal. This involves taking a Short Time Fourier Transform (STFT) and taking the sum of all frequencies bins which increase in energy from frame to frame. We ignore the decreasing bins because they are typically less rhythmically relevant.

$flux(t)$ is the raw spectral flux function. $rect(x)$ is the rectification function, such that $rect(x) = 0$ where $x < 0$, elsewhere $rect(x) = x$. X denotes the Fourier transform of a signal as a function of time and frequency.

$$flux(t) = \sum_{i=0}^{N} rect(|X(t,i)| - |X(t-1,i)|) \qquad (1)$$

Our OSS calculations are a slight modification of the Streamlined algorithm. The original approach involved using a low-passed copy of the result of the flux calculations to remove noise, but we use the unfiltered flux to preserve the exact timing of fluctuations, something more valuable for beat extraction than for global tempo analysis.

This approach gives a decent metric of where rhythmically salient events occur, however, it is flawed because it gives much less weight to low frequency fluctuations. By definition, low frequency signals fluctuate more slowly than high frequencies, and, as such, their fluctuations will have a much lower amplitude within a given STFT window. This means that this method is not able to clearly distinguish between a short click and bass drum, despite the fact that the bass drum is much more important for rhythmic perception.

In an attempt to remedy this, we make a copy of the flux using a downsampled copy of the input, and then mix it into the final output. We'll call this $flux_{adj}(t)$, as described below, with $a = 0.85$ and $b = 0.15$.

$$flux_{adj}(t) = a flux(t) + b flux_{downsampled}(t) \qquad (2)$$

Overall, this minor addition marginally improves results, but a better solution is worth investigating. Going forward, it may be worthwhile to use a multiband OSS calculation, as described by Bock, [18], or develop a different spectral flux calculation that compensates for bin fluctuation over frequency.

3.2 Peak Picking

Our next step is to extract each onset event with peak picking. We high-pass the OSS, which produces a new signal where sudden increases in flux are positive, and sudden decreases are negative. We segment the signal based on the positive zero crossings and find the maximal value for each region. We add these local maxima times to the set of peaks which gives us a set of discrete events that we can use for our final prediction.

The following expression defines the set of all positive high-passed OSS zero crossings:

$$\{cross_i = t \mid OSS_{hp}(t) > 0 \text{ and } OSS_{hp}(t-1) < 0\} \qquad (3)$$

$peak_i$ refers to the position of each onset event in time. The argmax function uses the last two parameters to specify a range.

$$peak_i = argmax(OSS_{hp}, cross_i, cross_{i+1}) \quad (4)$$

3.3 Beat Extraction

Next we must determine which beats to select. In the case that we have an estimate of the time delta between beats, we can use the error as a simple metric for determining if the peak is a beat. We get this expected beat time through induction by using a previous known beat's time plus some expected delta. Two initial beats are provided as input to the algorithm to begin the induction process. In our system, these are created by the user. Finding the peak with the minimum error is a trivial procedure on an ordered list of peaks; The error will increase monotonically, so once we find a peak with a positive error, no subsequent peaks will have a smaller error. This is the basic strategy for building our set of beats.

$\varepsilon(i, j)$ is the error for a given peak event, i, with the prior beat, j. If no peak is found within a certain error threshold, we skip a beat to find the next one.

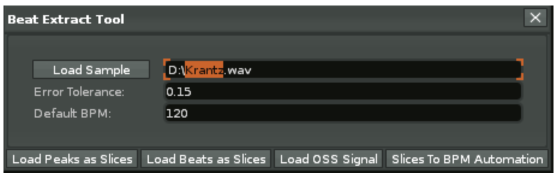
$$\varepsilon(i, j) = (peak_i - beat_{j-1}) - \delta_\mu(j) \quad (5)$$

In order to allow for varying tempos, we define a local beat delta, $\delta_\mu(i)$, which is expressed in terms of previously found beat deltas. $\delta_\mu(i)$ describes the expected beat delta for $beat_i$ as a function of previous beat deltas. Larger values of the averaging window, N, will result in more stable BPM fluctuations.

$$\delta_\mu(i) = \frac{\sum_{j=1}^{N} \delta_{i-j}}{N} \quad (6)$$

Finally we output this set of beats, which we will use to generate a tempo curve of the entire audio recording. An initial beat delta can either be supplied manually or predicted by an algorithm, such as the Streamlined algorithm. We also define an error tolerance, such that if no beat is found within that tolerance we double the length of the beat time and continue to search. When we do find a valid peak, we fill in the missing beats by subdividing the beat and reset the expected beat delta. Some optional parameters include the OSS threshold level, error tolerance and the averaging window, N, for the expected beat delta.

4. DAW INTEGRATION


Beat Extract Tool

Load Sample: D:\Krantz.wav
Error Tolerance: 0.15
Default BPM: 120

Figure 2. An image of the beat extraction tool made with the Renoise scripting API.

The Renoise DAW provides a scripting API that allows users to develop their own tools using the Lua scripting language. The API allows the developer to access some of the DAW's internal GUI elements and automate certain production tasks.

The tool that has been created to integrate this algorithm with Renoise takes advantage of Renoise's sample editor, which already has the ability to place slice markers that partition an audio file into multiple segments. The algorithm takes a sequence of beat times as inputs and generates a sequence as output.

To use the tool, the user can add two or more markers to initialize the algorithm, and then they can press the "Load Beats as Slices" button, which will run the algorithm on this input, generating a new set of slices. These generated slices can then be editted further, or they can be used to generate a tempo curve with the "Slices to BPM Automation" button. Extra beats can be added at various places in the audio file to correct errors, and ensure that the system correctly tracks erratic variations in tempo. These features constitute a user-in-the-loop interface, where the user can see the results of the algorithm, tweak the inputs, and then generate more refined output, as seen in figure 1.

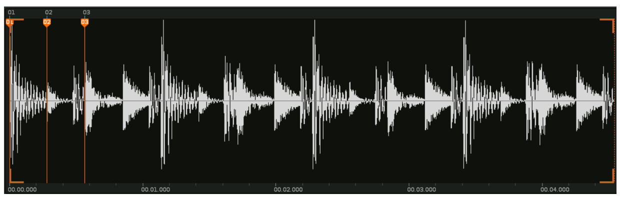


Figure 3. Renoise's sample editor. The orange lines are slice markers which the user can use to supply initial beat info.

There are a number of parameters that can be exposed to the user to fine tune the behaviour of the algorithm, however, they have not yet been integrated into the tool. These parameters include the confidence threshold, peak detection threshold, beat error tolerance and so on. This would make it easier to work with different kinds of material, as inputs that have more erratic rhythmic fluctuations will require more fine tuned parameters to produce accurate results.

The tool was tested with various recordings by the author, and it was found to work very well with music featuring strong percussion such as rock and jazz. However it was not suitable for use with less percussive recordings such as piano music. It may be useful in the future to study the effectiveness of the tool by comparing the time differences between manual and automatic beat annotation in a user study. This would give some more objective grounds to decide whether the algorithm is ready for practical use.

5. EVALUATION

The mir-eval[2] [5] evaluation suite was used to evaluate the efficacy of the algorithm against the beat tracking algorithms in the aubio [6], IBT [7], and BeatRoot [8] libraries.

[2] https://github.com/craffel/mir_eval

Algorithm	Aubio [6]	IBT [7]	BeatRoot [8]	Proposed (automatic)	Proposed (user initialized)
F-Measure	0.57	0.27	0.70	0.50	0.95

Table 1. F-Measure results from mir-eval. In the automatic column, the proposed algorithm is seeded with a bpm from the Streamlined tempo estimation algorithm, whereas in the user initialized column the algorithm is given two beats from the ground truth.

There are some complications when comparing this algorithm with other methods due to the fact that user input is used for initial BPM estimations, whereas other methods predict the BPM automatically. This puts the other algorithms at a disadvantage, and so it is not a fair comparison. Of the three methods tested, it was unclear if the host framework (Sonic Annotator) [19] provided the ability to supply a BPM estimate or initial beat. In order to show the difference in performance between the automatic and human-in-the-loop approach, the experimental results provided show our algorithm using user-supplied initial beat inputs (user initialized) as well as automatically generated input using the Streamlined tempo estimation algorithm (automatic). As seen in table 1, the algorithm performs much better when supplied with initial beats, bringing it's F-Measure of 0.50 up to 0.95.

The user supplied beat method uses two beats from the ground truth for each file in order to initialize the algorithm with a valid initial beat and initial beat delta. One concern with this approach is that mir-eval's F-measure score includes these shared beats, but their contribution to the score is negligible because they only make about 40 of the 1300 beats in the entire dataset. Since the ground truth annotations were derived from listeners tapping along to the audio, this should be a fair approximation of a real human-in-the-loop use case.

The data used includes 19 files from Sound and Music Computation for MIREX 2017, [20]. The set of files used were ones labelled as being easy; the files named SMC_271 through SMC_289. Each audio clip in the dataset is 40 seconds long and contains roughly 30 to 130 beats depending on the tempo. These files have strong percussive elements, while the remainder of the dataset is mostly classical music with weak rhythmic features. As stated previously the OSS calculation used is not adequate for detecting In the future this test data will need to be expanded, and the algorithm should be improved to work with less percussive audio sources.

The takeaway from these results is that a small amount of user provided information can be used to significantly improve the performance of a simple beat tracking algorithm. Other methods have been able to achieve 60%-80% F-measure accuracy, [21], [22], [12], and some methods that have been able to achieve up to 90% accuracy, [15]. This shows that the reframing of the beat induction problem allows a simplified algorithm to perform on par with, and even surpass, some of the most sophisticated algorithms within a limited practical context. These results are encouraging for further research into how these methods can be used to improve the functionality and usability of DAW software.

6. APPLICATIONS & FURTHER WORK

In this paper we have explored one of the more immediately obvious uses of beat tracking, which is synchronizing DAW software to an audio recording. There is, however, a wealth of other potential applications for beat tracking ranging from practical to experimental in nature.

Any given set of beats yields a tempo curve which is the function of tempo over time. The tempo curve is a signal like any other, and can be manipulated using filters and other conventional signal processing techniques. At the macro scale, the filtering of tempo curves can be used to remove tempo drift from recordings, as well as quantizing and performing other rhythmic corrections. With a high resolution tempo curve that captures rhythmic variation below the beat level, it would be possible to extract and manipulate musical rhythms in even more novel ways. One could imagine using a "tempo equalizer" on the tempo curve of a jazz recording to increase the amount of swing, or remove it entirely. The extraction of tempo curve information combined with time stretching and other audio manipulation techniques has many promising applications, and yet few, if any, of these applications have been realized in common audio software.

7. CONCLUSION

The results of this work show a promising potential for beat tracking algorithms in Digital Audio Workstations. We showed how simple beat tracking methods can be used to reliably synchronize DAW tempo with an audio source with user input and interactive corrections.

Currently the most significant concern with the method used in this work is the limited data used for testing. For future work we plan to conduct experiments with a larger and more varied data set. The algorithm also needs to be improved in order to deal with music that does not have strong percussive elements. Therefore, to make this viable, the most urgent issue to be addressed is the OSS calculation. The development of a new metric which is more able to capture rhythmic salience will be required for further work with material that does not contain strong transients.

Nevertheless, our method produces accurate results when provided with a small amount of user input, even surpassing the performance of conventional methods. This method is convenient and easy to use for production purposes in a DAW, and could easily be augmented with more sophisticated beat tracking techniques. With further work, DAW integrated beat tracking may become a powerful and interactive tool for music producers looking for creative ways to manipulate rhythm in digital music.

8. REFERENCES

[1] K. Jensen and T. H. Andersen, “Beat estimation on the beat,” in *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.* IEEE, 2003, pp. 87–90.

[2] D. P. Ellis, “Beat tracking by dynamic programming,” *Journal of New Music Research*, vol. 36, no. 1, pp. 51–60, 2007.

[3] J. L. Oliveira, M. E. Davies, F. Gouyon, and L. P. Reis, “Beat tracking for multiple applications: A multi-agent system architecture with state recovery,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 10, pp. 2696–2706, 2012.

[4] G. Percival and G. Tzanetakis, “Streamlined tempo estimation based on autocorrelation and cross-correlation with pulses,” *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 22, no. 12, pp. 1765–1776, 2014.

[5] C. Raffel, B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, D. P. Ellis, and C. C. Raffel, “mir_eval: A transparent implementation of common mir metrics,” in *In Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*. Citeseer, 2014.

[6] P. Brossier, “Automatic annotation of musical audio for interactive systems,” Ph.D. dissertation, Ph. D. thesis, Queen Mary University of London, London, UK, 2006.

[7] S. Dixon, “Automatic extraction of tempo and beat from expressive performances,” *Journal of New Music Research*, vol. 30, no. 1, pp. 39–58, 2001.

[8] C. Cannam, M. Mauch, M. E. P. Davies, S. Dixon, C. Landone, K. C. Noland, M. Levy, M. Zanoni, D. Stowell, and L. A. Figueira, “Mirex 2013 entry: Vamp plugins from the centre for digital music,” 2013.

[9] M. E. Davies and M. D. Plumbley, “Context-dependent beat tracking of musical audio,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 3, pp. 1009–1020, 2007.

[10] M. Goto, “An audio-based real-time beat tracking system for music with or without drum-sounds,” *Journal of New Music Research*, vol. 30, no. 2, pp. 159–171, 2001.

[11] A. Gkiokas, V. Katsouros, G. Carayannis, and T. Stafylakis, “Music tempo estimation and beat tracking by applying source separation and metrical relations.” in *ICASSP*, 2012, pp. 421–424.

[12] P. Grosche, M. Müller, and C. S. Sapp, “What makes beat tracking difficult? a case study on chopin mazurkas.” in *ISMIR*, 2010, pp. 649–654.

[13] S. Dixon and E. Cambouropoulos, “Beat tracking with musical knowledge,” in *ECAI*, 2000, pp. 626–630.

[14] S. Dixon, “Onset detection revisited,” in *Proceedings of the 9th International Conference on Digital Audio Effects*, vol. 120. Citeseer, 2006, pp. 133–137.

[15] ——, “Learning to detect onsets of acoustic piano tones,” in *Proceedings of the Workshop on Current Directions in Computer Music Research*, 2001, pp. 147–151.

[16] M. Goto and Y. Muraoka, “A real-time beat tracking system for audio signals.” in *ICMC*, 1995.

[17] S. Dixon, “An interactive beat tracking and visualisation system.” in *ICMC*. Citeseer, 2001.

[18] S. Böck, “Event detection in musical audio,” Ph.D. dissertation, PhD thesis. Johannes Kepler University Linz, Linz Austria, 2016.

[19] C. Cannam, M. O. Jewell, C. Rhodes, M. Sandler, and M. d’Inverno, “Linked data and you: Bringing music research software into the semantic web,” *Journal of New Music Research*, vol. 39, no. 4, pp. 313–325, 2010.

[20] A. Holzapfel, M. E. Davies, J. R. Zapata, J. L. Oliveira, and F. Gouyon, “Selective sampling for beat tracking evaluation,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 9, pp. 2539–2548, 2012.

[21] A. Pikrakis, I. Antonopoulos, and S. Theodoridis, “Music meter and tempo tracking from raw polyphonic audio.” in *ISMIR*, 2004.

[22] A. Mottaghi, K. Behdin, A. Esmaeili, M. Heydari, and F. Marvasti, “Obtain: Real-time beat tracking in audio signals,” *arXiv preprint arXiv:1704.02216*, 2017.